## Contents

*Chapter 1 • Accessing Data*  ................................................................. 1

- Introduction .................................................................................. 2
- Terms to Be Familiar With ............................................................ 2
- Common Tasks for Accessing and Manipulating Data ....................... 3
- Load a Client-Side File ................................................................. 4
- Load a Server-Side File ............................................................... 5
- Load a Database Table ............................................................... 12
- Save an In-Memory Table ............................................................ 13
- Drop an In-Memory Table ........................................................... 14
- Delete a File from a Caslib’s Data Source ..................................... 15
- Data Compression ........................................................................ 15
Chapter 1
Accessing Data

Introduction ................................................................. 2
Terms to Be Familiar With .............................................. 2
Common Tasks for Accessing and Manipulating Data ...................... 3
Load a Client-Side File ....................................................... 4
Prerequisites ............................................................... 4
Example ................................................................. 4
Key Ideas .............................................................. 5
See Also ............................................................... 5
Load a Server-Side File ..................................................... 5
Prerequisites ............................................................... 5
Example ................................................................. 5
Results ................................................................. 8
Key Ideas .............................................................. 11
See Also ............................................................... 12
Load a Database Table .................................................... 12
Prerequisites ............................................................... 12
Example ................................................................. 12
Key Ideas .............................................................. 13
Save an In-Memory Table ................................................. 13
Example ................................................................. 13
Key Ideas .............................................................. 14
Drop an In-Memory Table .................................................. 14
Example ................................................................. 14
Key Ideas .............................................................. 15
Delete a File from a Caslib’s Data Source .................................. 15
Example ................................................................. 15
Key Ideas .............................................................. 15
Data Compression .......................................................... 15
Overview of Data Compression ........................................... 15
Compressed Tables and the DATA Step .................................... 16
Compressed Tables and the CASUTIL Procedure ......................... 17
Performance Considerations ............................................. 17
Interactions ............................................................ 17
Introduction

SAS Viya analytical procedures and some DATA step features use input data from in-memory tables on SAS Cloud Analytic Services only. This document takes you through common tasks for loading and accessing your input data with SAS Cloud Analytic Services. The methods that these tasks use for loading and accessing are:

- caslibs
- CASUTIL procedure
- DATA step

You can use SAS/CONNECT to transfer remote data sources directly into in-memory tables. However, SAS/CONNECT is a separately licensed product. For more information about SAS/CONNECT, see SAS/CONNECT User’s Guide.

Terms to Be Familiar With

Before we begin, here are a few important terms:

active caslib
your session must have a default location for server-side data access. This is the active caslib. The term "active caslib" is used rather than default caslib because the caslib that your session uses is modified as caslibs are added and dropped.

caslib
the mechanism for accessing data with SAS Cloud Analytic Services. At its simplest, a caslib provides access to files in a data source, such as a database or file system directory, and to in-memory tables.

data connector
a data connector is the software that is used with a caslib to read server-based data sources like databases and Hive. There are a few data connectors for file-based caslibs. These data connectors are used to control reading data files such as setting the file encoding.

file
the source data that is in a caslib’s data source. For a caslib that uses a path-based data source, this is natural. For a caslib that uses a database as a data source, the tables in the database are referred to as files.

session
when you initially connect to SAS Cloud Analytic Services, your session is started on the server. Data access and communication is performed through the session. Your programs communicate with the session to request actions. Many sessions can operate concurrently, actions execute serially within a session. In most cases, programmers start and use one session only.

table
is used to refer to in-memory data. After a file (using the preceding definition) is loaded into the server, it is referred to as a table.
## Common Tasks for Accessing and Manipulating Data

### Table 1.1  Common Tasks for Accessing Data

<table>
<thead>
<tr>
<th>Task</th>
<th>Sample Syntax</th>
</tr>
</thead>
</table>
| Load a SAS data set.  
*Tip:* This is a good learning step if you are familiar with SAS and want to learn how SAS libraries, data sets, and SAS Cloud Analytic Services work together. | ```
proc casutil;
load data=libref.member-name  
casout="table-name";
run;
``` |
| Load a client-side data file. | ```
proc casutil;
load file="/path/to/file.suffix"  
casout="table-name";
run;
``` |
| List caslibs. This shows you the server-side data sources that SAS Cloud Analytic Services can access. | `caslib _all_ list;` |
| Add a file-based caslib.  
*Tip:* Remember that the specified PATH= must be accessible from the host for the SAS Cloud Analytic Services controller. | ```
caslib datasourcename=(srctype="path")  
path="/data01";
``` |
| Determine the data files in a caslib that the server can access. | ```
proc casutil;
list files incaslib="name";
run;
``` |
| Load a server-side data file. | ```
proc casutil;
load casdata="file-name.suffix"  
casout="table-name";
run;
``` |
| Load a table from database. | ```
proc casutil;
load casdata="database-table-name"  
casout="table-name";
run;
``` |
| List in-memory tables. | ```
proc casutil;
list tables incaslib="name";
run;
``` |
| Assign a CAS engine libref and bind it to a caslib. This is how you access tables with a SAS procedure or the DATA step. | `libname mycas cas caslib="name";` |
Load a Client-Side File

Prerequisites

The following example assumes that you have a Microsoft Excel file. The sample code assumes that a file named /data/WorldData.xlsx is available in the Server Files and Folders section of SAS Studio.

Example

This example shows how to load data from a SAS library and how to load data from a Microsoft Excel file. This approach is appropriate for smaller tables and ad hoc data analysis.

```sas
   caslib hps datasource=(srctype="hdfs")                          /* 1 */
       path="/hps";
   proc casutil incaslib="hps" outcaslib="hps";                     /* 2 */
   load data=sashelp.iris promote;                                  /* 3 */
       contents casdata="iris";
   load file="/data/WorldData.xlsx" casout="worlddata";             /* 4 */
       contents casdata="worlddata";
   run;
```

1  Add a caslib to access the /hps directory in HDFS. By default, adding a caslib sets it as the active caslib.

2  The CASUTIL procedure statement includes the INCASLIB= and OUTCASLIB= options. This is a best practice to ensure that tables are read from the caslib that you expect and are saved to the caslib that you expect.

3  The DATA= argument indicates that the table is transferred from the SAS client host to SAS Cloud Analytic Services. Replace the Sashelp.Iris value with a libref and table that you want to use. The PROMOTE option makes the Iris table a global-scope table and available to other sessions that use the Hps caslib. Use the CONTENTS statement to ensure that the table includes the column names and data types that you expect.

4  The FILE= argument indicates that the file is a client-side file that is accessible to SAS and not to SAS Cloud Analytic Services. The file, WorldData.xlsx, is transferred to the server and then imported with a table name of Worlddata. This LOAD statement does not include the PROMOTE option, so the in-memory Worlddata table can be accessed only from the same session.
Key Ideas

- The LOAD DATA= and LOAD FILE= statements in the CASUTIL procedure are used for accessing client-side data.
- The CONTENTS statement is used to display information such as column names and data types.
- By default, when you add a caslib, that caslib becomes the active caslib. Use the NOTACTIVE option to add a caslib without making it active.

See Also

- “CASUTIL Procedure” in SAS Cloud Analytic Services: Language Reference
- “CASLIB Statement” in SAS Cloud Analytic Services: Language Reference

Load a Server-Side File

Prerequisites

The following example assumes the following:

- You can create a small CSV file in the file system that is associated with the directory for your personal caslib.

Example

This example shows how to access two server-side files and load the data into CAS:

- a CSV file that describes the performance of a toy catapult. The first line of the file does not contain column names. The example shows how to specify names.
- a Microsoft Excel file.

For the CSV file, a description for the data is shown in the example. The values are as follows:

5,10,10,11,10,11,3
5.5,16,3,16,1,15,6
6,23,0,18,7,20,5
6.5,23,3,28,6,26,0
7,27,3,25,10,23,1

The program is as follows:

```cas
cas casauto sessopts=(caslib="casuser");
```
libname mycas cas caslib="casuser"; /* 2 */

/* first, load the data from the CSV file */
proc casutil incaslib="casuser" outcaslib="casuser";
  contents casdata="catapult.csv"; /* 3 */
  load casdata="catapult.csv" casout="catapultraw"
    importoptions=(filetype="csv" /* 4 */
      encoding="latin1"
      getnames="false"
    vars={
      (name="turns", label="Number of turns", type="double"),
      (name="first_ft", label="Feet for first try", type="double"),
      (name="first_in", label="Inches for first try", type="double"),
      (name="second_ft", label="Feet for second try", type="double"),
      (name="second_in", label="Inches for second try", type="double"),
      (name="third_ft", label="Feet for third try", type="double"),
      (name="third_in", label="Inches for third try", type="double")
    }
  ) replace;
  save casdata="catapultraw" replace; /* 5 */
  contents casdata="catapultraw.sashdat";
quit;

data mycas.catapult (promote=yes) / sessref=casauto; /* 6 */
  set mycas.catapultraw;
  first  = 12 * first_ft  + first_in;
  second = 12 * second_ft + second_in;
  third  = 12 * third_ft  + third_in;
run;

proc casutil incaslib="casuser";
  contents casdata="catapult"; /* 7 */
  droptable casdata="catapultraw"; /* 8 */
quit;

/* simple scatter plot */
proc sgplot data=mycas.catapult; /* 9 */
  scatter x=turns y=first;
  scatter x=turns y=second;
  scatter x=turns y=third;

  /*--X Axis--*/
  xaxis grid label="Number of turns";

  /*--Y Axis--*/
  yaxis grid label="Distance, in inches";
run;

/* second, load the Excel file */
proc casutil incaslib="casuser" outcaslib="casuser";
  list files; /* 10 */
  contents casdata="historicalcpi.xls";
The SESSOPTS= option is used with the CASLIB= session option to ensure that the Casuser personal caslib is set as the active caslib.

The CAS engine LIBNAME statement assigns the Mycas libref and binds it to the Casuser caslib.

The CONTENTS statement shows the file information and column information for the CSV file. See Figure 1.1 on page 8.

The CASDATA= argument indicates that the file is read from the caslib's data source. The IMPORTOPTIONS= specify how to read the file.

The SAVE statement makes a copy of the imported data as a SASHDAT file. This is part of the data life cycle. If the file is imported correctly, then subsequent analyses of the data can begin from the SASHDAT file. The CONTENTS statement shows that the column names and labels are applied. See Figure 1.2 on page 8.

The DATA step is used to combine each set of foot and inch measures into a single column. The PROMOTE= option is used to make the table available to other sessions that you start. The SESSREF= option is used to ensure that the DATA step runs in CAS.

The last CONTENTS statement is used to display the table information, table details, and column information for the in-memory table. See Figure 1.3 on page 9.

The DROPTABLE statement is used to free the memory resources that are used for the data from the CSV file. The copy of the data that was made with the SAVE statement is not deleted, only the in-memory resources are freed.

The SGPLOT procedure uses the Mycas CAS engine libref. When Mycas was assigned at the start of the program, the CASLIB= option bound it to the Casuser caslib. This ensures that the libref always accesses tables in that caslib.

The LIST FILES statement is used to list the files in the caslib's data source. In this case, the personal caslib, Casuser, uses the OS file system. See Figure 1.4 on page 9.

The LOAD CASDATA= argument specifies the Historicalcpi.xls file. The LABEL= option is used to specify a description of the data.
Results

The following display shows the results of the CONTENTS statement. Notice that the anticipated column names match the first line of the CSV file. This is corrected in the subsequent LOAD CASDATA= statement when the GETNAMES= option is set to false.

Figure 1.1 CONTENTS Statement Results for the CSV File

The CASUTIL Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>catapult.csv</td>
<td>-rw-r--r--</td>
<td></td>
<td></td>
<td>0.1KB</td>
<td>01Feb2016:14:50:36</td>
</tr>
</tbody>
</table>

The following display applies to the second CONTENTS statement, after the LOAD CASDATA= statement that include the IMPORTOPTIONS= settings. Notice that the column names and labels are applied.

Figure 1.2 CONTENTS Statement Results for the SASHDAT File

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>catapultraw.sashdat</td>
<td>-rwxr-xr-x</td>
<td></td>
<td>24.2KB</td>
<td>01Feb2016:19:27:20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>turns</td>
<td>Number of turns</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_ft</td>
<td>Feet for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_in</td>
<td>Inches for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_ft</td>
<td>Feet for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_in</td>
<td>Inches for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_ft</td>
<td>Feet for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_in</td>
<td>Inches for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>
**Figure 1.3** CONTENTS Statement Results for the In-Memory Table

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NLS encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Duplicated Rows</th>
<th>View</th>
<th>Source Name</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATAPULT</td>
<td>5</td>
<td>10</td>
<td>utf-8</td>
<td>01Feb2016:19:29:25</td>
<td>01Feb2016:19:27:55</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>_T_SAAD153D_7FF_65328EBA</td>
<td>No</td>
</tr>
</tbody>
</table>

**Figure 1.4** LIST FILES Statement Results for a Path-Based Caslib

<table>
<thead>
<tr>
<th>Node</th>
<th>Number of Blocks</th>
<th>Active Blocks</th>
<th>Rows</th>
<th>Fixed Data Size</th>
<th>Variable Data Size</th>
<th>Blocks Mapped</th>
<th>Memory Mapped</th>
<th>Blocks Unmapped</th>
<th>Memory Unmapped</th>
<th>Blocks Allocated</th>
<th>Memory Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>400</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**The CASUTIL Procedure**

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>catapult.csv</td>
<td>-rw-r--r--</td>
<td></td>
<td></td>
<td>0.1KB</td>
<td>01Feb2016:14:50:35</td>
</tr>
<tr>
<td>historicalpts.xls</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>56.5KB</td>
<td>02Feb2016:19:08:19</td>
</tr>
<tr>
<td>mycas.distinct.sashdat</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>7.2KB</td>
<td>12Nov2015:16:48:12</td>
</tr>
<tr>
<td>cars_large_part.sashdat</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>20.4MB</td>
<td>26Jan2016:12:44:45</td>
</tr>
</tbody>
</table>
Figure 1.5  Scatter Plot Results for the Catapult Table
The following display shows the results of the CONTENTS statement for the Historicalcpi.xls file. By default, the column names are read from a file.

**Figure 1.6 CONTENTS Statement Results for the XLS File**

### The CASUTIL Procedure

#### File Information for historicalcpi.xls in caslib CASUSER( ).

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>historicalcpi.xls</td>
<td>rw-r-xr-x</td>
<td></td>
<td></td>
<td>56.5KB</td>
<td>02Feb2016:19:08:19</td>
</tr>
</tbody>
</table>

#### Column Information for historicalcpi.xls in Caslib CASUSER( )

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer price indexes historical data, 1974 through 2014</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>B</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>C</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>D</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>E</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>F</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AO</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AP</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AQ</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AR</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AS</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AT</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
</tbody>
</table>

**Key Ideas**

- Reading files from a caslib's data source is the most efficient way to access data. One key to recognizing that data is read from a caslib's data source is the presence of the CASDATA= argument.

- The CONTENTS statement in the CASUTIL procedure can display information for files, tables, and columns. To view information about a file, specify the filename, including the suffix in the CASDATA= option. After the table is loaded into memory, you drop the suffix or use the table name that you specified in the CASOUT= option.

- You can specify IMPORTOPTIONS= to describe how to load the data. For delimited files, the common options are to specify the file encoding and whether to get column names from the first line of the file.

- After the data for a file is imported, save a copy as a SASHDAT file.
Load a Database Table

Prerequisites

The following example assumes the following:

- You are granted access to data in the data source.
- You know the connection information such as host, port, and so on.
- Your SAS Cloud Analytic Services installation is licensed and configured to use the client software for the data source vendor that you want to access. For installation-time configuration information, see SAS Viya for Linux: Deployment Guide.

Example

Loading tables into the server from a caslib’s data source is the most efficient way to load data. In this example, a table is read from Oracle and the in-memory table is kept in the same caslib.

caslib oralib datasource=(/* 1 */
   srctype="oracle",
   uid="DBUSER",
   pwd="secret",
   path="//dbserver.example.com:1521/dbname",
   schema="DBUSER"
);

caslib oralib datasource=(/* 1 */
   srctype="oracle",
   uid="DBUSER",
   pwd="secret",
   path="//dbserver.example.com:1521/dbname",
   schema="DBUSER"
);

proc casutil;
   list files; /* 2 */
droptable casdata="sales" quiet;
   contents casdata="sales"; /* 4 */
   load casdata="sales" casout="sales" promote /* 5 */
      label="Fact table for User-to-Item Analysis"
      varlist={
         (name="USERID" label="User ID"),
         (name="ITEMID" label="Item ID")
      };
   contents casdata="sales"; /* 6 */
quit;

See Also

- “CASUTIL Procedure” in SAS Cloud Analytic Services: Language Reference
- “CASLIB Statement” in SAS Cloud Analytic Services: Language Reference
- “DATA Step Basics” in SAS Cloud Analytic Services: DATA Step Programming
Add a caslib that uses Oracle as the data source. Oralib becomes the active caslib for the session and the subsequent programming statements use it for input and output.

The LIST FILES statement displays the tables that are available in the Oracle database.

The DROPTABLE statement includes the QUIET option. Running this statement is useful on repeated runs because it ensures that no table named Sales can be in-memory to interfere with the subsequent LOAD CASDATA= statement.

Because the first CONTENTS statement follows the DROPTABLE statement, this ensures that the table information and column information from Oracle are read.

The CASDATA= argument in the LOAD statement indicates that the Sales table is read from the caslib's data source (Oracle) into SAS Cloud Analytic Services. Options are specified to add labels to the table and columns.

Because the last CONTENTS statement follows the LOAD statement, table information and column information is displayed for the in-memory copy of the Sales table that was read from Oracle.

**Key Ideas**

- The **CASLIB** statement adds a server-side data source to SAS Cloud Analytic Services.
- In this example, the active caslib is Oralib. Remember that when you add a caslib, by default, it becomes the active caslib.
- For information about data source connection parameters, see “Data Connectors” in *SAS Cloud Analytic Services: Language Reference*.

---

**Save an In-Memory Table**

**Example**

This example demonstrates the following:

- saving a table to caslib named Hps that uses HDFS as a data source. The table is saved as a SASHDAT file.
- saving a table from a caslib that uses Oracle to a caslib named Hps that uses HDFS as a data source. The table is saved as a SASHDAT file.

```
proc casutil incaslib="hps" outcaslib="hps";
    load casdata="customers.sashdat" casout="customers";
run;

/* From some other application, or a DATA step, the */
/* Customers table is modified with a change that     */
/* is important to save.                           */
proc casutil incaslib="hps" outcaslib="hps";
    save casdata="customers" replace;            /* 2 */
```
run;

caslib orsales datasource=(
   srctype="oracle"
   username="DBUSER"
   password="secret"
   path="//dbserver.example.com:1521/sales"
   schema="DBUSER"
);

proc casutil incaslib="orsales";
   load casdata="sales" casout="sales"; /* 2 */
   save casdata="sales" outcaslib="hps";
run;

1 The SAVE statement uses the Hps caslib from the OUTCASLIB= procedure statement option. The next time a LOAD statement is used with the Customers table, the table includes the changes. You can use the CASOUT= option to specify an alternative filename other than the default, customers.sashdat.

2 The LOAD statement reads the table named Sales from the Orsales caslib that uses Oracle Database as a data source. To save a copy of the table in HDFS, the SAVE statement uses the OUTCASLIB= option to specify the Hps caslib that uses HDFS as a data source.

**Key Ideas**

• You can save in-memory tables as SASHDAT files or CSV files in path-based caslibs only. If you need to save a table from a caslib that is not path-based, then you can specify the OUTCASLIB= option.

• For information about data source connection parameters to use in the CASLIB statement, see “Data Connectors” in *SAS Cloud Analytic Services: Language Reference*.

---

### Drop an In-Memory Table

**Example**

When you drop an in-memory table, only the in-memory table is affected. If the table was loaded from a caslib's data source, the table in the data source is unaffected.

```
proc casutil incaslib="hps" outcaslib="hps";
   load casdata="sales.sashdat" casout="sales"; /* 3 */
run;

proc casutil incaslib="hps";
   droptable casdata="sales"; /* 2 */
run;
```
1. The LOAD statement loads a file named Sales.sashdat from the Hps caslib.
2. The DROPTABLE statement drops the in-memory table.

**Key Ideas**

- If you drop a session-scope table, then only the session that loaded the table is affected.
- If you drop a global-scope table, then the table might be accessed from multiple sessions. The table is dropped after any actions that access the table are complete.
- Be aware that dropping a global-scope table can affect other sessions if the actions that are run by other sessions expect the table to be in memory.

---

**Delete a File from a Caslib’s Data Source**

**Example**

When you delete a file from a data source, it does not affect an in-memory copy. The term "file" refers to a file in a caslib with a path-based data source or a table in a caslib with a server-based data source.

The following example demonstrates deleting a file named Sales from the data source associated with the Hps caslib.

```sas
proc casutil;
   deletesource casdata="sales.sashdat" incaslib="hps";
run;
```

**Key Ideas**

- You can delete files from path-based caslibs only.
- Include the filename suffix in the CASDATA= argument.

---

**Data Compression**

**Overview of Data Compression**

SAS Cloud Analytic Services supports and performs all compression for in-memory tables. When you transfer a table to the server and request compression, rows are sent to the server as is and the server compresses them.

All data in a row, both character and numeric variables, are compressed. Every row in a table is compressed. The server does not support some rows in compressed form and others as uncompressed.
For matrices of computed doubles (those with many decimal places), compression might not reduce the storage requirements at all. For rows with many long character variables that consist mostly of blanks, the compression ratio can be very high. For rows with mixed variables, where most doubles do not have fractional parts and most character variables have a small amount of blank padding, the compression ratio is typically moderate. As with most cases of using compression, character variables tend to compress the most and the ratio depends on your data.

**Compressed Tables and the DATA Step**

This example shows how to use the COMPRESS= data set option for SAS Cloud Analytic Services.

**Example Code 1.1  Creating a Compressed Table with a DATA Step**

```sas
libname mycas cas host="cloud.example.com" port=5570;

data mycas.prdsale (compress=yes);
  set sashelp.prdsale;
run;
```

After the table is loaded into memory, you can access the compressed table with the mycas.prdsale table reference.

SAS Cloud Analytic Services supports the APPEND= data set option for compressed tables. This example shows how to add new (uncompressed) rows to the compressed table.

**Example Code 1.2  Appending Rows to a Compressed Table**

```sas
data mycas.prdsale (append=yes);
  somelib.newrows;
run;
```

Because the mycas.prdsale table is already compressed, the new rows are automatically compressed as they are appended to the table. Specifying COMPRESS= with APPEND= has no effect. If the table is compressed, the server compresses the new rows. If the table is not compressed, then the server does not compress the new rows even if you specify COMPRESS=YES. The compressed or uncompressed state of the table determines how rows are appended.

Partitioning and compression are supported together. This example creates a new in-memory table that is partitioned and compressed.

**Example Code 1.3  Creating a Partitioned and Compressed Table**

```sas
data mycas.iris (partition=(species) compress=yes);
  set sashelp.iris;
run;

data mycas.iris (append=yes);
  set somelib.moreirises;
run;
```

In the first DATA step statement, the Iris data set is loaded into memory on the server. The table is partitioned by the formatted values of the Species variable. The table is also compressed. In the second DATA step statement, the table is appended to with more rows. Because the in-memory table is already partitioned and compressed, the new rows are automatically partitioned and compressed when they are appended.
Compressed Tables and the CASUTIL Procedure

You can use the CASUTIL procedure to load data in memory on SAS Cloud Analytic Services.

```sas
proc casutil;
  load data=sashelp.prdsale casout="prdsale" compress;
quit;
```

This example uses the COMPRESS option to read the Prdsale data set from the sashelp library and compress it in-memory on the server. Be aware that you must specify the COMPRESS option for each table that you want to load in compressed form.

When you read SASHDAT tables into memory, compression depends on these factors for the resulting in-memory tables:

- whether a WHERE clause is used
- whether the SASHDAT table is compressed on disk

If you specify a WHERE clause when loading a SASHDAT file with compression, the server uncompresses the rows as it evaluates the WHERE clause. This results in an uncompressed in-memory table. The memory efficiencies of the SASHDAT table format are forfeited in this scenario because the server had to apply the WHERE clause.

If you do not specify a WHERE clause, the server ignores the COMPRESS option and relies on whether the SASHDAT file is compressed. If the SASHDAT file is compressed, the in-memory representation of the table is also compressed. If the SASHDAT file is not compressed, then neither is the in-memory representation. The server ignores the option so that it can keep the memory efficiencies of the SASHDAT file format: When a SASHDAT table is loaded in memory, the in-memory representation is identical to the on-disk representation.

Performance Considerations

Compression exchanges less memory use for more CPU use. It slows down any request that processes the data. An in-memory table consists of blocks of rows. When the server works with a compressed table, the blocks of rows must be uncompressed before the server can work with the variables. In some cases, a request can require five times longer to run with a compressed table rather than an uncompressed table.

For example, if you want to summarize two variables in a table that has 100 variables, all 100 columns must be uncompressed in order to locate the data for the two variables of interest. If you specify a WHERE clause, then the server must uncompress the data before the WHERE clause can be applied. Like the example where only two of 100 variables are used, if the WHERE clause is very restrictive, then there is a substantial performance penalty to filter out most of the rows.

Working with SASHDAT tables that are loaded from HDFS is the most memory-efficient way to use the server. Using compressed SASHDAT tables preserves the memory efficiencies, but still incurs the performance penalty of uncompressing the rows as the server operates on each row.

Interactions

Here are the interactions for compressed tables and SAS programs.

- You can use a compressed table in programs like any other table.
You can define calculated columns for compressed tables with the TEMPNAMES= data set option.

You can append to compressed tables. This is also supported for compressed tables that have partitioning with or without ordering.

See Also

- “APPEND= Data Set Option” in SAS Cloud Analytic Services: Language Reference
- “COMPRESS= Data Set Option” in SAS Cloud Analytic Services: Language Reference
- “ORDERBY= Data Set Option” in SAS Cloud Analytic Services: Language Reference
- “TEMPNAMES= Data Set Option” in SAS Cloud Analytic Services: Language Reference
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.